Correction Handling Exceptions - 0

B - Throwable

Correction Handling Exceptions - 1

B - FileNotFoundException

ClassCastException extends RuntimeException (unchecked exception),

FileNotFoundException extends IOException, IOException extends Exception (checked exception),

ExceptionInInitializerError is from Error family and is thrown by an static initializer block,

Correction Handling Exceptions - 2

D - Compilation error

Even though an instance of FileNotFoundException is thrown by method m1() at runtime, but method m1() declares to throw IOException.

Reference variable s is of Super type and hence for compiler, call to s.m1(); is to method m1() of Super, which throws IOException.

And as IOException is checked exception hence calling code should handle it.

Correction Handling Exceptions - 3

B - None of the System.out.printiIn statements are executed

main(args) method is invoked recursively without specifying any exit condition, so this code ultimately throws java.lang.StackOverflowError. StackOverflowError is a subclass of Error type and not Exception type, hence it is not handled. Stack trace is printed to the console and program ends abruptly.

Java doesn’t allow to catch specific checked exceptions if these are not thrown by the statements inside try block.

Correction Handling Exceptions - 4

B - No, HELLO is not printed on the console

Correction Handling Exceptions - 5

C - Compilation error at Line 3

If a method declares to throw Exception or its sub-type other than RuntimeException types, then calling method should follow handle or declare rule. In this case, as method m1() declares to throw Exception, so main method should either declare the same exception or its super type in its throws clause OR m1(); should be surrounded by try-catch block.

Correction Handling Exceptions - 6

B - Compilation error.

Method m1() throws Exception (checked) and it declares to throw it, so no issues with method m1().

But main() method neither provides catch handler nor throws clause and hence main method gives Compilation error.

Handle or Declare rule should be followed for checked exception if you are not re-throwing it.

Correction Handling Exceptions - 7

A - throw

catch is for catching the exception and not throwing it.

thrown is not a java keyword.

throws is used to declare the exceptions a method can throw.

To manually throw an exception, throw keyword is used. e.g., throw new Exception();

Correction Handling Exceptions - 8

A - NumberFormatException

B - IllegalArgumentException

C - ClassCastException

ClassCastException, NumberFormatException and IllegalArgumentException are Runtime exceptions.

There are no exception classes in java with the names: NullException and ArrayIndexException.

Correction Handling Exceptions - 9

C -

```
public class C4 implements I1 {
  public void m1() throws Exception{}
}
```

NOTE: Question is asking for “incorrect” implementation and not “correct” implementation.

According to overriding rules, if super class / interface method declares to throw a checked exception, then overriding method of sub class / implementer class has following options:

1. May not declare to throw any checked exception,

2. May declare to throw the same checked exception thrown by super class / interface method,

3. May declare to throw the sub class of the exception thrown by super class / interface method,

4. Cannot declare to throw the super class of the exception thrown by super class / interface method

Correction Handling Exceptions - 10

D - class Test gives compilation error

FileNotFoundException extends IOException and hence catch block of FileNotFoundException should appear before the catch block of IOException.

Therefore, class Test causes compilation error.

Correction Handling Exceptions - 11

A - A is printed to the console, stack trace is printed and then program ends abruptly.

Method m1() throws an instance of ArithmeticException and method m1() doesn’t handle it, so it forwards the exception to calling method main.

Method main doesn’t handle ArithmeticException so it forwards it to JVM, but just before that finally block is executed. This prints A on to the console.

After that JVM prints the stack trace and terminates the program abruptly.

Correction Handling Exceptions - 12

A - No, HELLO is not printed on to the console

To invoke the special main method, JVM loads the class in the memory. At that time, static fields of Test class are initialized. d1 is of Double type so null is assigned to it.

x is also static variable so d1.intValue(); is executed and as d1 is null hence d1.intValue() throws a NullPointerException and as a result an instance of java.lang.ExceptionInInitializerError is thrown.

Correction Handling Exceptions - 13

D - Exception

Method m1() throws an instance of TestException, which is a checked exception as it extends Exception class.

So in throws clause we must provide:

1. Checked exception.
2. Exception of TestException type or it’s super types (Exception, Throwable), Object cannot be used in throws clause.

Correction Handling Exceptions - 14

E - Compilation error

Java doesn’t allow to catch specific checked exceptions if these are not thrown by the statements inside try block.

catch(FileNotFoundException ex) {} causes compilation error in this case as System.out.println(1); will never throw FileNotFoundException.

NOTE: Java allows to catch Exception type. catch(Exception ex) {} will never cause compilation error.

Correction Handling Exceptions - 15

A - Derived: m1()

NullPointerException extends RuntimeException, but there are no overriding rules related to unchecked exceptions.

So, method m1() in Derived class correctly overrides Base class method.

Rest is simple polymorphism. obj refers to an instance of Derived class and hence obj.m1(); invokes method m1() of Derived class, which prints “Derived: m1()” to the console.

Correction Handling Exceptions - 16

B - Compilation error

class Error extends Throwable, so `obj instanceof Error;` and `obj instanceof Throwable;` return true.

But Error class is not related to Exception and RuntimeException classes in multilevel inheritance and that is why Line n1 and Line n2 causes compilation error.

Correction Handling Exceptions - 17

D - XZ

Even though method m1() declares to throw IOException but at runtime an instance of FileNotFoundException is thrown.

A catch handler for FileNotFoundException is available and hence X is printed on to the console.

After that finally block is executed, which prints Z to the console.

Correction Handling Exceptions - 18

E - Program terminates abruptly

for(;;) is an infinite loop and hence `sb.append("OCA");` causes OutOfMemoryError which is a subclass of Error class.

main(String []) method throws OutOfMemoryError and program terminates abruptly.

Correction Handling Exceptions - 19

A - Replace Line 4 with static void print() throws Exception {

B - Surround Line 11 with below try-catch block:

```
try {
  ReadTheFile.print();
} catch(Exception e) {
  e.printStackTrace();
}
```

This question is tricky as 2 changes are related and not independent. Let's first check the reason for compilation error. Line 5 throws a checked exception, IOException but it is not declared in the throws clause. So, print method should have throws clause for IOException or the classes in top hierarchy such as Exception or Throwable.

Based on this deduction, Line 4 can be replaced with either "static void print() throws Exception {" or "static void print() throws Throwable" but we will have to select one out of these as after replacing Line 4, Line 11 will start giving error as we are not handling the checked exception at Line 11.

This part is easy, do we have other options, which mention "Throwable"? NO. Then mark the first option as "Replace Line 4 with static void print() throws Exception {".

As, print() method throws Exception, so main method should handle Exception or its super type and not it's subtype. Two options working only with IOException can be ruled out.

Multi-catch statement "catch(IOException | Exception e)" gives compilation error as IOException and Exception are related to each other in multilevel inheritance. So you are left with only one option to pair with our 1st choice:

Surround Line 11 with below try-catch block:

```
try {
    ReadTheFile.print();
} catch(Exception e) {
    e.printStackTrace();
}
```

Correction Handling Exceptions - 20

B - NullPointerException is thrown at runtime

Variable arr is a class variable of int [] type, so by default it is initialized to null.

In if block, arr.length > 0 is checked first. Accessing length property on null reference throws NullPointerException.

Correct logical if block declaration should be:

if(arr != null && arr.length > 0)

First check for null and then access properties/methods.

Correction Handling Exceptions - 21

C - Compilation error

NullPointerException extends RuntimeException and in multi-catch syntax we can’t specify multiple Exceptions related to each other in multilevel inheritance.

Correction Handling Exceptions - 22

F - Class1.java and Class2.java compile successfully

Method declaring checked exception in its throws clause doesn’t mean that it should have code to actually throw that type of Exceptions.

So even though read() method of Class1 declares to throw FileNotFoundException but its body doesn’t actually throw an instance of FileNotFoundException.

Variable and method name can be same as class name, so code of Class2 is also valid.

Remember: Though you don’t get any compilation error but it is not recommended to use the Class name for variable and method names.

LOCAL variable can be declared with final modifier only.

msg variable inside print() method of Class3 is declared private and this causes compilation error.

Correction Handling Exceptions - 23

B - Yes, HELLO is printed on to the console

To invoke the special main method, JVM loads the class in the memory. At that time, static fields of Test class are initialized. d1 is of Double type so null is assigned to it.

x is not static variable, so int x = d1.intValue(); is not executed. Class is loaded successfully in the memory and “HELLO” is printed on to the console.

NOTE: new Test() will throw NullPointerException but not ExceptionInInitializerError.

Correction Handling Exceptions - 24

C - Compilation error

NullPointerException extends RuntimeException and in multi-catch syntax we can’t specify multiple Exceptions related to each other in multilevel inheritance.

Correction Handling Exceptions - 25

F - Class1.java and Class2.java compile successfully

Method declaring checked exception in its throws clause doesn’t mean that it should have code to actually throw that type of Exceptions.

So even though read() method of Class1 declares to throw FileNotFoundException but its body doesn’t actually throw an instance of FileNotFoundException.

Variable and method name can be same as class name, so code of Class2 is also valid.

Remember: Though you don’t get any compilation error but it is not recommended to use the Class name for variable and method names.

LOCAL variable can be declared with final modifier only.

msg variable inside print() method of Class3 is declared private and this causes compilation error.

Correction Handling Exceptions - 26

B - Yes, HELLO is printed on to the console

To invoke the special main method, JVM loads the class in the memory. At that time, static fields of Test class are initialized. d1 is of Double type so null is assigned to it.

x is not static variable, so int x = d1.intValue(); is not executed. Class is loaded successfully in the memory and “HELLO” is printed on to the console.

NOTE: new Test() will throw NullPointerException but not ExceptionInInitializerError.

Correction Handling Exceptions - 27

A - An exception is thrown at runtime

Line n1 creates an array instance of ILogger containing 2 elements. null is assigned to both the array elements. Line n1 compiles successfully.

As, log() method is declared in ILogger interface, hence statement at Line n2: logger.log(); doesn’t cause any compilation error. Compiler is happy to see that log() method is invoked on the reference variable of ILogger type.

1st iteration:

logger –> null, logger.log(); throws NullPointerException as method log() is invoked on null reference.

Correction Handling Exceptions - 28

D - Test class executes successfully and prints: DEER JUMPS DEER JUMPS TO 5 FEET

Method jump() in Animal class declares to throw RuntimeException.

Overriding method may or may not throw any RuntimeException. Only thing to remember is that if overridden method throws any unchecked exception or Error, then overriding method must not throw any checked exceptions. Line n1 compiles successfully as it correctly overrides the jump() method of Animal class.

Class Deer also provides overloaded jump(int) method.

Inside main(String []) method, reference variable ‘animal’ is of Animal class (supertype) and it refers to an instance of Deer class (subtype), this is polymorphism and allowed in Java.

As instance is of Deer class, hence ‘animal’ reference can easily be casted to Deer type. Line n2 and Line n3 compiles successfully and on execution prints below on to the console:

DEER JUMPS DEER JUMPS TO 5 FEET

Correction Handling Exceptions - 29

D - CAUGHT SUCCESSFULLY is printed on to the console and program terminates successfully

Even though it seems like method m() will not compile successfully, but starting with JDK 7, it is allowed to use super class reference variable in throw statement referring to sub class Exception object.

In this case, method m() throws SQLException and compiler knows that variable e (Exception type) refers to an instance of SQLException only and hence allows it.

Program executes successfully and prints CAUGHT SUCCESSFULLY on to the console.

Correction Handling Exceptions - 30

D - None of the other options

search(String) method declares to throw FileNotFoundException, which is a checked exception. It returns true if match is found otherwise it throws an instance of FileNotFoundException.

main(String[]) provides try-catch block around `search("virat.pdf")` and catch handler checks for FileNotFoundException. Given code compiles successfully.

There are 4 elements in ‘names’ array, so starting index is 0 and end index is 3, but given for loop goes till index number 4.

As search string is “virat.pdf” (not present in names array), hence for loop will execute for i = 0, 1, 2, 3, 4.

For i = 4, `names[i].equalsIgnoreCase(name)` throws ArrayIndexOutOfBoundsException (it is a RuntimeException). main(String[]) method doesn’t provide handler for ArrayIndexOutOfBoundsException and therefore stack trace is printed on to the console and program terminates abruptly.

Correction Handling Exceptions - 31

D - Program ends abruptly

Classes in Exception framework are normal java classes, hence null can be used wherever instances of Exception classes are used, so Line 7 compiles successfully.

No issues with Line 12 as method availableSeats() declares to throw SQLException and main(String []) method code correctly handles it.

Program compiles successfully but on execution, NullPointerException is thrown, stack trace is printed on to the console and program ends abruptly.

If you debug the code, you would find that internal routine for throwing null exception causes NullPointerException.

Correction Handling Exceptions - 32

C - Compilation error

throw ex; causes compilation error as div method doesn’t declare to throw Exception (checked) type.

Correction Handling Exceptions - 33

D - FINALLY is printed to the console, stack trace is printed and then program ends abruptly

As method div() doesn’t declare to throw any Checked Exception, hence main(String []) method is not suppose to handle it, try-finally without catch is valid here. There is no compilation error in the code.

Method div() throws an instance of ArithmeticException and method div() doesn’t handle it, so it forwards the exception to calling method main(String []).

Method main(String []) doesn’t handle ArithmeticException so it forwards it to JVM, but just before that, finally block is executed. This prints FINALLY on to the console.

After that JVM prints the stack trace and terminates the program abruptly.

Correction Handling Exceptions - 34

B - public void travel() throws RuntimeException {}

C - public abstract void travel();

E - public void travel(Object obj) {}

F - abstract void travel();

Both Traveller and BeachTraveller are abstract classes and BeachTraveller extends Traveller. It is possible to have abstract class without any abstract method. Code as is compiles successfully as BeachTraveller inherits travel(String) method of Traveller class.

But as per the question, /*INSERT*/ must be replaced such that there is no compilation error.

Let’s check all the options one by one:

abstract void travel(); ✓ This is method overloading. BeachTraveller has 2 methods: `void travel(String){}` and `abstract void travel()`.

abstract void travel(String beach); ✓ As BeachTraveller is abstract, hence travel(String) method can be declared abstract.

public abstract void travel(); ✓ This is method overloading. BeachTraveller has 2 methods: `void travel(String){}` and `abstract void travel()`.

public void travel() throws RuntimeException {}: ✓ This is method overloading. BeachTraveller has 2 methods: `void travel(String){}` and `public void travel() throws RuntimeException {}`.

public void travel(String beach) throws Exception {}: ✗ As overridden method doesn’t declare to throw any checked Exception hence overriding method is not allowed to declare to throw Exception.

void travel(String beach) throws java.io.IOException {} ✗ As overridden method doesn’t declare to throw any checked Exception hence overriding method is not allowed to declare to throw java.io.IOException.

public void travel(Object obj) {} ✓ This is method overloading. BeachTraveller has 2 methods: `void travel(String){}` and `public void travel(Object){}`.

Correction Handling Exceptions - 35

A -

```
try {
  save();
  log();
} catch(SQLException | |OException ex) {}
```

E -

```
try {
  save();
  log();
} catch(IOException | SQLException ex) {}
```

F -

```
try {
  save();
  log();
} catch(Exception ex) {}
```

save() method throws IOException (which is a Checked Exception) and log() method throws SQLException (which is also a Checked Exception).

Let’s check all the options one by one (I am just using the catch-block as try-block of all the options are same):

catch(IOException | SQLException ex) {}: ✓ As IOException and SQLException are not related to each other in multi-level inheritance, hence this multi-catch syntax is valid.

catch(SQLException | IOException ex) {}: ✓ Same as above, order of exceptions in multi-catch syntax doesn’t matter.

catch(IOException | Exception ex) {}: ✗ Causes compilation error as IOException extends Exception.

catch(SQLException | Exception ex) {}: ✗ Causes compilation error as SQLException extends Exception.

catch(Exception | RuntimeException ex) {}: ✗ Causes compilation error as RuntimeException extends Exception.

catch(Exception ex) {}: ✓ As Exception is the super class of both IOException and SQLException, hence it can handle both the exceptions.

Correction Handling Exceptions - 36

E - INHALE-EXHALE-INHALE-EXHALE

As command-line argument is not passed, hence Line n1 throws ArrayIndexOutOfBoundsException (subclass of RuntimeException), handler is available in inner catch block, it executes Line n1 and prints INHALE- on to the console.

throw e; re-throws the exception.

But before exception instance is forwarded to outer catch-block, inner finally-block gets executed and prints EXHALE- on to the console.

In outer try-catch block, handler for RuntimeException is available, so outer catch-block gets executed and prints INHALE- on to the console.

After that outer finally-block gets executed and prints EXHALE- on to the console.

Hence, the output is: INHALE-EXHALE-INHALE-EXHALE

Correction Handling Exceptions - 37

A - Test class executes successfully and prints HAKUNAMATATA on to the console

It is legal for the constructors to have throws clause.

Constructors are not inherited by the Child class so there is no method overriding rules related to the constructors but as one constructor invokes other constructors implicitly or explicitly by using this(…) or super(…), hence exception handling becomes interesting.

Java compiler adds super(); as the first statement inside Child class’s constructor:

Child() throws Exception {

```
super(); //added by the compiler

System.out.println("MATATA");
```

}

super(); invokes the constructor of Parent class (which declares to throw IOException), but as no-argument constructor of Child class declares to throw Exception (super class of IOException), hence IOException is also handled. There is no compilation error and output is: HAKUNAMATATA

Correction Handling Exceptions - 38

A - RuntimeException

B - NullPointerException

C - SQLException

D - SQLWarning

G - Error

At Line n1, reference variable ‘obj’ is of Multiplier type (supertype) and it refers to an instance of Calculator class (subtype). This is polymorphism and allowed in Java.

multiply(int…) method declared in Multiplier interface declares to throw SQLException, hence the catch handler for Line n1 should provide handler for SQLException or its supertype. As catch-handler for SQLException is available, therefore Test class compiles successfully.

According to overriding rules, if super class / interface method declares to throw a checked exception, then overriding method of sub class / implementer class has following options:

1. May not declare to throw any checked exception
2. May declare to throw the same checked exception thrown by super class / interface method: SQLException is a valid option.
3. May declare to throw the sub class of the exception thrown by super class / interface method: SQLWarning is a valid option.
4. Cannot declare to throw the super class of the exception thrown by super class / interface method: Exception, Throwable are not valid options.
5. Cannot declare to throw unrelated checked exception: java.io.IOException is not a valid option as it is not related java.sql.SQLException in multi-level inheritance.
6. May declare to throw any RuntimeException or Error: RuntimeException, NullPointerException and Error are valid options.

Therefore 5 options can successfully replace /*INSERT*/: SQLException, SQLWarning, RuntimeException, Error and NullPointerException

Correction Handling Exceptions - 39

D - Line 11 causes compilation failure

Exception is a java class, so `e = null;` is a valid statement and compiles successfully.

If you comment Line 10, and simply throw e, then code would compile successfully as compiler is certain that ‘e’ would refer to an instance of SQLException only.

But the moment compiler finds `e = null;`, `throw e;` (Line 11) causes compilation error as at runtime ‘e’ may refer to any Exception type.

NOTE: No issues with Line 17 as method checkData() declares to throw SQLException and main(String []) method code correctly handles it.

Correction Handling Exceptions - 40

A - AE is printed on to the console and program terminates successfully

Any RuntimeException can be thrown without any need it to be declared in throws clause of surrounding method.

`throw (RuntimeException)e;` doesn’t cause any compilation error.

Even though variable ‘e’ is type casted to RuntimeException but exception object is still of ArithmeticException, which is caught in main method and ‘AE’ is printed to the console.

Correction Handling Exceptions - 41

C - INNER FINALLY 1 FINALLY 2

`System.out.println(1/0);` throws ArithmeticException, handler is available in inner catch-block, it executes and prints “INNER” to the console.

Once an exception is handled, no other catch block will get executed unless the exception is re-thrown.

Inner finally-block gets executed and prints “FINALLY 1” to the console.

Rule is finally-block always gets executed, so outer finally-block gets executed and prints “FINALLY 2” to the console.

Correction Handling Exceptions - 42

D - Test class executes successfully and prints CARPE DIEM on to the console

It is legal for the constructors to have throws clause.

Constructors are not inherited by the Sub class so there is no method overriding rules related to the constructors but as one constructor invokes other constructors implicitly or explicitly by using this(...) or super(...), hence exception handling becomes interesting.

Java compiler adds super(); as the first statement inside Sub class's constructor:

```
Sub() throws IOException {
    super(); //added by the compiler
    System.out.println("DIEM");
}
```

super(); invokes the constructor of Super class (which declares to throw RuntimeException), as RuntimeException is unchecked exception, therefore no handling is necessary in the constructor of Sub class.

Sub class's constructor declares to throw IOException but main(String []) method handles it.

There is no compilation error and output is: CARPE DIEM

Correction Handling Exceptions - 43

A - TRAVEL is printed on to the console and program terminates successfully

According to overriding rules, if super class / interface method declares to throw a checked exception, then overriding method of sub class / implementer class has following options:

1. May not declare to throw any checked exception.
2. May declare to throw the same checked exception thrown by super class / interface method.
3. May declare to throw the sub class of the exception thrown by super class / interface method.
4. Cannot declare to throw the super class of the exception thrown by super class / interface method.
5. Cannot declare to throw unrelated checked exception.
6. May declare to throw any RuntimeException or Error.

default methods were added in Java 8 and TravelBlogger class correctly overrides the default method blog() of Blogger interface. Blogger interface compiles successfully.

At Line n1, ‘blogger’ is of Blogger type (supertype) and it refers to an instance of TravelBlogger class (subtype), this is polymorphism and allowed in Java. Line n1 compiles successfully.

At Line n2, blog() method is being invoked on typecasting ‘blogger’ to TravelBlogger and as TravelBlogger class doesn’t declare to throw any checked exception, hence Line n2 compiles successfully.

As instance is of TravelBlogger type, therefore on execution, Line n2 invokes blog() method of TravelBlogger instance, which prints TRAVEL on to the console.

Correction Handling Exceptions - 44

A - Program ends abruptly

Classes in Exception framework are normal java classes, hence null can be used wherever instances of Exception classes are used, so Line 10 compiles successfully.

No issues with Line 16 as method getReport() declares to throw SQLException and main(String []) method code correctly handles it.

Program compiles successfully but on execution, NullPointerException is thrown, stack trace is printed on to the console and program ends abruptly.

If you debug the code, you would find that internal routine for throwing null exception causes NullPointerException.

Correction Handling Exceptions - 45

E - Compilation error

Both try and catch blocks have return; statement, which means either of the return statements will definitely get executed. Hence, compiler tags `System.out.println("DONE");` as unreachable and this causes compilation error.

Correction Handling Exceptions - 46

D - Compilation error

Method test() throws Exception (checked) and it declares to throw it, so no issues with method test().

But main(String []) method neither provides catch handler nor throws clause and hence main(String []) method causes compilation error.

Handle or Declare rule should be followed for checked exception if you are not re-throwing it.

Correction Handling Exceptions - 47

B - Compilation error only in Derived class

It is legal for the constructors to have throws clause.

Constructors are not inherited by the Derived class so there is no method overriding rules related to the constructors but as one constructor invokes other constructors implicitly or explicitly by using this(...) or super(...), hence exception handling becomes interesting.

Java compiler adds super(); as the first statement inside Derived class's constructor:

```
Derived() throws FileNotFoundException {
    super(); //added by the compiler
    System.out.print(2);
}
```

As super(); invokes the constructor of Base class (which declares to throw IOException), compiler complains as Derived class no-argument constructor doesn't declare to throw IOException. It declares to throw FileNotFoundException (subclass of IOException), which is not enough for the instances of IOException.

Correction Handling Exceptions - 48

D - Derived: log()

NullPointerException extends RuntimeException. Overriding method may or may not throw any RuntimeException. Only thing to remember is that if overridden method throws any unchecked exception or Error, then overriding method must not throw any checked exceptions.

So, method log() in Derived class correctly overrides Base class's method.

Rest is simple polymorphism. 'obj' refers to an instance of Derived class and hence obj.log(); invokes method log() of Derived class, which prints "Derived: log()" on to the console.

Correction Handling Exceptions - 49

E - Replace Line 14 with ‘catch(RuntimeException e) {’

Throwable is the root class of the exception hierarchy and it contains some useful constructors:

1. public Throwable() {...} : No-argument constructor
2. public Throwable(String message) {...} : Pass the detail message
3. public Throwable(String message, Throwable cause) {...} : Pass the detail message and the cause
4. public Throwable(Throwable cause) {...} : Pass the cause

Exception and RuntimeException classes also provide similar constructors.

Throwable class also contains methods, which are inherited by all the subclasses (Exception, RuntimeException etc.)

1. public String getMessage() {...} : Returns the detail message (E.g. detail message set by 2nd and 3rd constructor)
2. public String toString() {} :

Returns a short description of this throwable. The result is the concatenation of:

the name of the class of this object

“:” (a colon and a space)

the result of invoking this object’s getLocalizedMessage() method

If getLocalizedMessage() returns null, then just the class name is returned.

In multi-catch statement, classes with multi-level hierarchical relationship can’t be used.

RuntimeException is subclass of Exception, IllegalArgumentException is indirect subclass of Exception and IllegalArgumentException is subclass of RuntimeException, hence these pairs can’t be used in multi-catch statement.

Only one option is left to replace Line 14 with ‘catch(RuntimeException e) {’.

Commenting out Line 14, Line 15 and Line 16 will resolve the compilation error but it will print the whole stack trace rather than just printing the message.

Correction Handling Exceptions - 50

C - abstract List get() throws ArrayIndexOutOfBoundsException;

Few things to keep in mind:

1.

There are 2 rules related to return types of overriding method:

A. If return type of overridden method is of primitive type, then overriding method should use same primitive type.

B. If return type of overridden method is of reference type, then overriding method can use same reference type or its sub-type (also known as covariant return type).

2.

In case of overriding, if overridden method declares to throw any RuntimeException or Error, overriding method may or may not throw any RuntimeException but overriding method must not throw any checked exceptions.

3.

In generics syntax, Parameterized types are not polymorphic, this means even if B is subtype of A, List **is not subtype of List. Remember this point. So below syntaxes are NOT allowed:**

List list = new ArrayList(); OR ArrayList list = new ArrayList();

Let’s check all the options one by one:

abstract List get() throws ArrayIndexOutOfBoundsException; => ✓ It returns the same return type ‘List’ and it is allowed to throw any RuntimeException (ArrayIndexOutOfBoundsException is RuntimeException)

abstract List get(); => ✗ List is not subtype of List, it is not covariant return type.

abstract ArrayList get() throws Exception; => ✗ As overridden method declares to throw IndexOutOfBoundsException, which is a Runtime Exception, overriding method is not allowed to declare to throw any checked Exception. Class Exception and its subclasses are checked exceptions.

abstract ArrayList get(); => ✗ ArrayList is not subtype of List, it is not covariant return type.

Correction Handling Exceptions - 51

C - Method `getData()` causes compilation error

If you don’t initialize variable e inside catch block using `e = new SQLException();` and simply throw e, then code would compile successfully as compiler is certain that ‘e’ would refer to an instance of SQLException only.

But the moment compiler finds `e = new SQLException();`, `throw e;` causes compilation error as at runtime ‘e’ may refer to any Exception type.

Correction Handling Exceptions - 52

B - Yes

Throwable is the root class of the exception hierarchy and it contains some useful constructors:

1. public Throwable() {...} : No-argument constructor
2. public Throwable(String message) {...} : Pass the detail message
3. public Throwable(String message, Throwable cause) {...} : Pass the detail message and the cause
4. public Throwable(Throwable cause) {...} : Pass the cause

Exception and RuntimeException classes also provide similar constructors.

Hence all 3 statements Line n1, Line n2 and Line n3 compile successfully.

Throwable class also contains methods, which are inherited by all the subclasses (Exception, RuntimeException etc.)

1. public String getMessage() {...} : Returns the detail message (E.g. detail message set by 2nd and 3rd constructor)
2. public String toString() {} :

Returns a short description of this throwable. The result is the concatenation of:

the name of the class of this object

“:” (a colon and a space)

the result of invoking this object’s getLocalizedMessage() method

If getLocalizedMessage returns null, then just the class name is returned.

Because of the toString() method,

Line n1 prints “java.lang.RuntimeException”.

Line n2 prints “java.lang.RuntimeException: HELLO”

Line n3 prints “java.lang.Exception: java.lang.RuntimeException: HELLO”

Correction Handling Exceptions - 53

E - Compilation error

java.io.FileNotFoundException exception is a checked exception.

Java doesn’t allow to catch specific checked exceptions if these are not thrown by the statements inside try block. catch(FileNotFoundException ex) {} causes compilation error in this case as System.out.println(1); will never throw FileNotFoundException.

NOTE: Java allows to catch Exception type. catch(Exception ex) {} will never cause compilation error.